Effects Of Pipelining On Algorithms For The MC68020

by David Olsen, Lockheed, Sunnyvale, CA

In the past, the performance of an algorithm on a microprocessor was determined by how fast the processor ran and how many instructions the algorithm required. This has been changed by the advent of pipelining found in today's 32-bit microprocessors. Now, the order in which instructions are executed, location of the instruction on an even or odd boundary in memory and the form of the algorithm affect the performance.

The execution of instruction can be broken down into six distinct phases: instruction fetch, instruction decode, effective address calculation, operand fetching, execution and storage of data. In simpler microprocessors, all these stages will be completed before execution of the next instruction is started. The pipelining in the 68020 allows for the starting of these phases on the following instructions before the completion of the current instruction. The 68020 has three units which are responsible for these functions, the pipeline, the bus controller and the sequencer.

The pipeline has three stages; the first stage holds an operand, and the second and third stages hold an operand or an operand extension word (**Figure 1**). An operand extension word is the second word of a two-word instruction. For example, the instruction "MOV %FFFF, D1" shows the holding of intermediate data. The pipeline is responsible for decoding the instructions and supplying the sequencer with any operand extension words. The bus controller is responsible for all activity on the address and data bus, including instruction fetch, instruction prefetch, operand fetching, data fetching and storage of data. The sequencer is responsible for the effective address calculation, that is, determining bus address for all the addressing modes and activity involving the ALU.

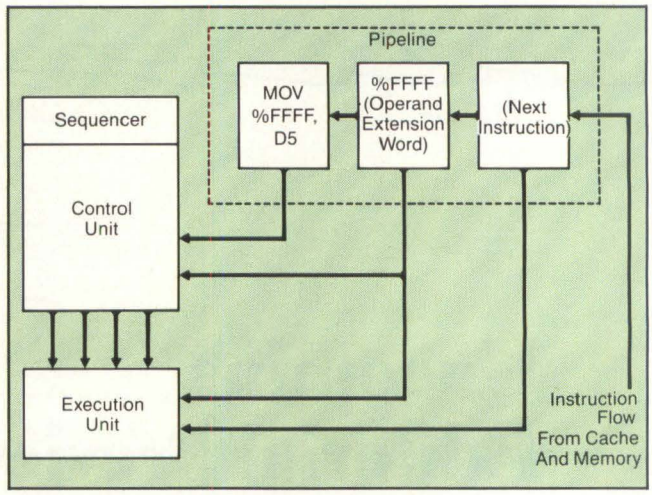

Figure 1: The pipeline of the 68020 has three stages. The first stage holds an operand, and the second and third stages hold an operand or operand extension word.

The 68020 differs from the 68000 in that these three units perform independently. In the 68000, the bus controller can prefetch instructions when not needed to fetch or to store operands. In the 68020, the sequencer has a degree of independence. If the

David Olsen is Associate Engineer Senior at Lockheed Space Systems Division, Sunnyvale, CA.

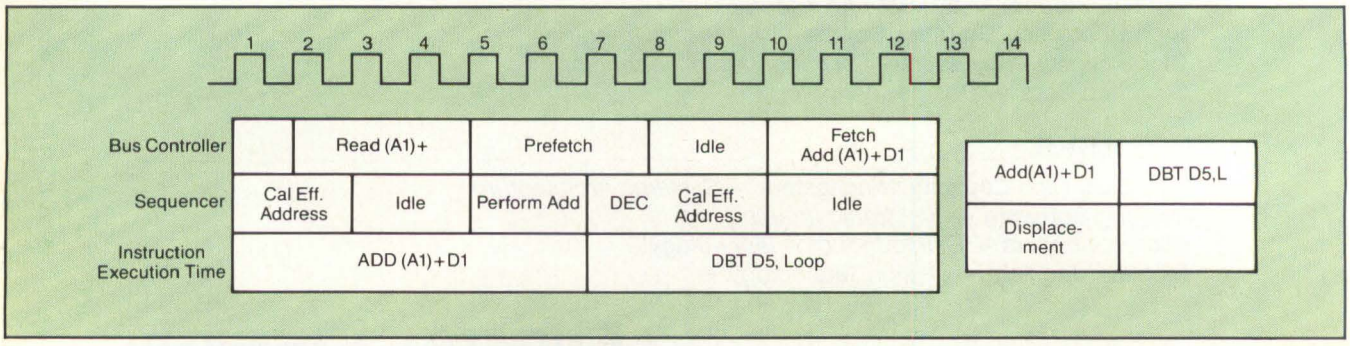

Figure 2: Timing diagram for a simple algorithm to add an array of numbers.

sequencer is finished with its required phase of an instruction, and the bus controller is finishing the last phase of instruction execution as in MOV D1,(A1), the sequencer can then look at the next instruction prefetched into the pipeline and start execution. The overlap of the bus controller finishing the last phase of an instruction and the sequencer starting its execution phase of a new instruction can be great enough that the sequencer would finish an instruction at the same time that the bus controller finishes the previous instruction. Of course, there are locks in the pipeline to prevent instruction dependent sequences from occurring out of order.

The performance of today's μPs is being affected by the increasing use of pipelining.

The effect of pipelining can be illustrated by a simple algorithm for adding up an array of numbers. The most obvious way of implementing this algorithm is as follows (D5 contains the length of the array):

```
LOOP: ADD (A1)+,D1   ┐
                     ├ Main Loop
      DBT D5,LOOP    ┘
```

The timing for these instructions running on the 68020 is shown in **Figure 2**. The times the sequencer and the bus controller are active are shown. The sequencer is idle during the first ADD instruction, clock cycle 4. This is because the ADD D1,D2 instruction requires the data be fetched from memory, (A), and loaded into register D1 before it can add D1 to D2. Therefore, the sequencer must wait for the bus controller to complete before performing the addition.

The first algorithm requires 12 clock cycles to sum an element in an array. However, to maximize the instruction overlap, this algorithm can be rewritten as follows:

```
LOOP: MOV (A1)+,D1   ┐
      ADD D3,D4      │
      MOV (A2)+,D3   ├ Main Loop
      ADD D1,D2      │
      DBT D5,LOOP    ┘
      ADD D4,D2
```

The time associated with this algorithm is shown in **Figure 3**. The array is broken in half, and each half is summed separately. During the first instruction when the sequencer is finished with its task or calculating the effective address and incrementing the address register A1, the sequencer can start executing the next instruction in the pipeline (ADD D1,D2). This is at the end of clock cycle 4. Because the second instruction, ADD D3,D4, does not require the data being fetched by the first MOV instruction, there is no interlock between the sequencer and the bus controller; that is, the sequencer can continue processing instructions in the pipeline. This is called an instruction independent sequence. Compare the first two instructions of each algorithm and notice that for the first algorithm the sequencer must wait for the data being fetched before being able to continue processing instructions in the pipeline. The second algorithm eliminated idle sequencer time. By alternating the additions, the idle sequencer time, which occurs when the ADD (A1)+,D1 instruction is waiting for data to be fetched, is eliminated. The sequencer is never idle.

Performing two additions takes 18 cycles, or 9 cycles per addition of array element. This is a savings of 3 cycles, or 67% of the time required in the first algorithm. However, this requires more memory and initiation and uses two extra registers, which might require being restored if used in a subroutine. However, an array size larger than 10-15 elements is the approximate crossover point for improved performance.

One criterion for using this type of algorithm separation is that the algorithm must be highly repetitive, a loop. Also, the calculations within the repetitive part must be associative; i.e., the order in which they occur does not matter or the elements being operated on are independent of each other. It does not matter what order an array is summed or what order an array is searched for an element.

The performance of today's microprocessors is being affected by the increasing use of pipelining. The firmware programmer must gain familiarity with the effects of pipelining and how it will enable him to increase the peformance of the system. For the person involved in the selection of a high level language for the microprocessor where high performance is important, the idea of pipelining must be kept in mind. New compilers will undoubtedly take advantage of the pipelining, and optimizing compilers will become available which have pipeline reorganizers. DD

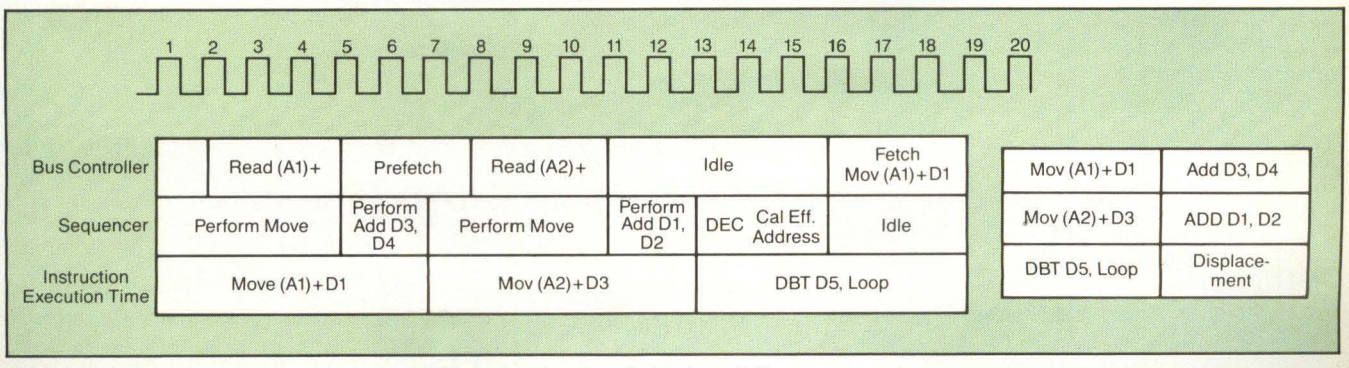

Figure 3: Timing diagram for the rewritten algorithm showing the elimination of idle sequencer time.